



universität
wien

Fakultät für Informatik



@csunivie

Breeding Unicorns: Developing Trustworthy and Scalable Randomness Beacons

Samvid Dharanikota, Michael Jensen, Sebastian Kristensen, Mathias Michno, Yvonne-Anne Pignolet, Rene Hansen, Stefan Schmid





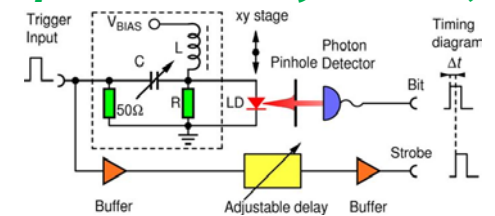
Randomness: An Important Tool of Our (Digital) Society

- In *algorithm* design:
 - Faster algorithms through randomization
 - Symmetry breaking in distributed algorithms
 - Secure cryptographic protocols based on random seeds (e.g., elliptic curves)
- In entertainment:
 - *Lotteries, games* (Roulette)
 - Drafts in sports
- In politics:
 - Military *drafts*, assignment of kids to schools
- *Blockchain. more soon*

Randomness: An Important Tool of Our (Digital) Society

- In *algorithm* design:
 - Faster algorithms through randomization
 - Symmetry breaking in distributed algorithms
 - Secure cryptographic protocols based on random seeds (e.g., elliptic curves)
- In entertainment:
 - *Lotteries, games* (Roulette)
 - Drafts in sports
- In politics:
 - Military *drafts*, assignment of kids to schools
- *Blockchain*. more soon

Can be solved by
producing
randomness locally
myself
(e.g.,
/dev/urandom,
using quantum
photonics, etc.)





Randomness: An Important Tool of Our (Digital) Society

- In *algorithm* design:
 - Faster algorithms through randomization
 - Symmetry breaking in distributed algorithms
 - Secure cryptographic protocols based on random seeds (e.g., elliptic curves)
- In entertainment:
 - *Lotteries, games* (Roulette)
 - Drafts in sports
- In politics:
 - Military *drafts*, assignment of kids to schools
- *Blockchain*. more soon



**Requires randomness that
is trustworthy for
everybody! Output of
„shared randomness“ is
relevant to multiple
stakeholders.**

Randomness: An Important Tool of Our (Digital) Society

- In *algorithm* design:
 - Faster algorithms through randomization
 - Symmetry breaking in distributed algorithms
 - Secure cryptographic protocols based on random seeds (e.g., elliptic curves)
- In entertainment:
 - *Lotteries, games* (Roulette)
 - Drafts in sports
- In politics:
 - Military *drafts*, assignment of kids to schools
- *Blockchain*. more soon

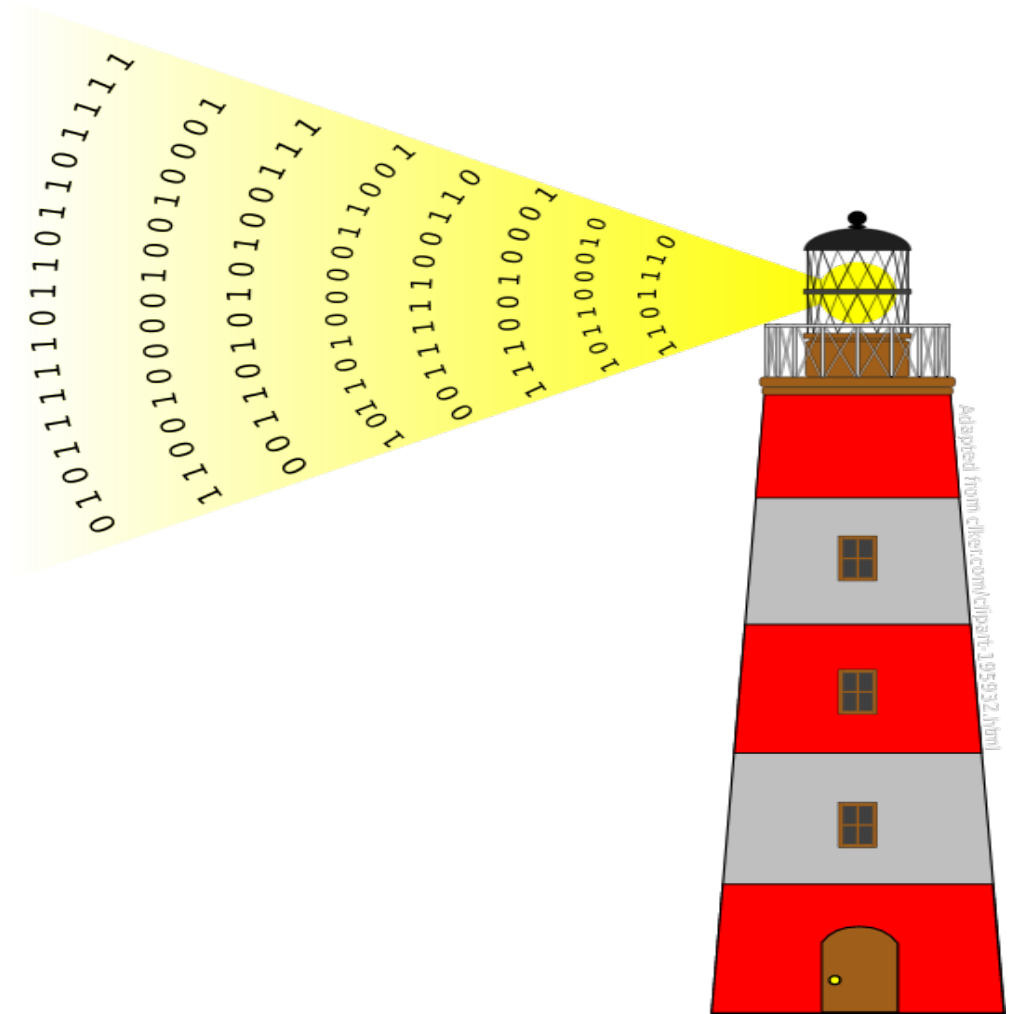


How to realize such trustworthy shared randomness?

“*Output of randomness*” is relevant to multiple stakeholders.

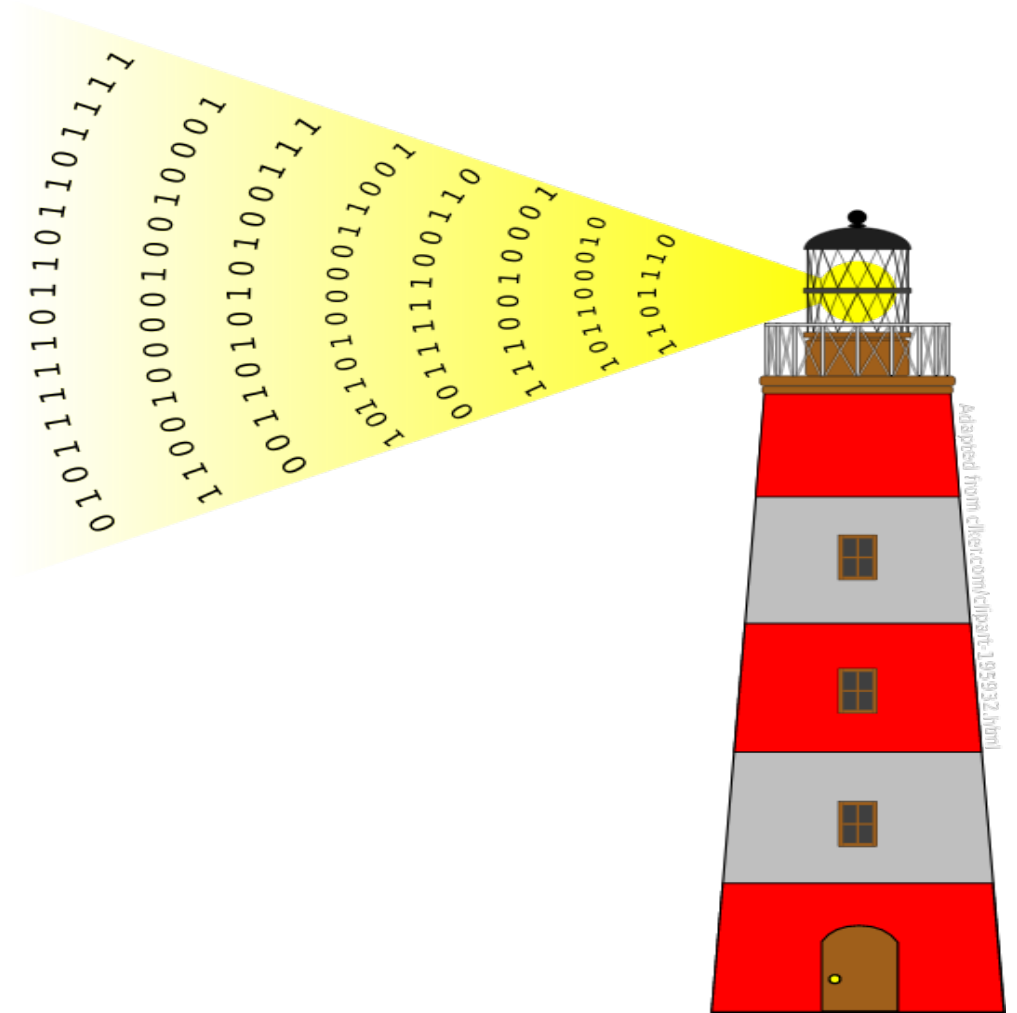
Vision: Randomness Beacon

- Vision: service emitting *unpredictable random values*...
- *Public*. seen by „everybody“



Vision: Randomness Beacon

- Vision: service emitting *unpredictable random values*...
- *Public*. seen by „everybody“
- Introduced by *Michael O. Rabin* in 1983





When Are Randomness Beacons Useful?

- When group of users need to agree on some random value but do *not trust each other*
- E.g.:
 - Lotteries
 - Secure *elections*
 - Prevent *selfish mining*
 - Blockchain/distributed systems' *consensus* mechanisms
 - Overcoming slow bootstrapping in zero-knowledge proofs
 - Rabin's example: providing *security to e-mail* based protocols *with minimal trust*

When Are Randomness Beacons Useful?

- When group of users need to agree on some random value but do *not* *reach other*
- E.g.:
 - Lotteries
 - Secure *elections*
 - Prevent *spoofing*
 - Blockchain *consensus* mechanisms
 - Overcoming *bootstrapping* in zero-knowledge proofs
 - Rabin's example: providing *security to e-mail* based protocols *with minimal trust*

Trustworthy randomness: "A tool of democracy"



Available today?!

- E.g.: NIST's randomness beacon
- One new value per minute



Certificate

NIST Beacon 1.0 used an X.509 certificate with the Federal Common Policy CA as the ultimate root authority for some of the records. The certificate is available [here](#)

The Beacon Signing Key changed in May of 2017. Not all records will successfully validate.

Viewer

The example application below uses the REST API described above to navigate the data.

To visit an arbitrary time value please click on the input control containing the date and time value to display a calendar to select the desired value.

First Start Chain Previous Next Last

12/26/2018 5:07 pm

Beacon Record

Version: Version 1.0

Frequency: 60 seconds

Time: 12/26/2018 5:07 pm (1545840420)

Seed Value: C2E5D63FA9B8276C1E4568BAE5B50C242856707519C295AEC71B99E72A0588CF
EE2D79A7BC7ADD71D8BAC87FF753730D8DBDC171CE2DF6D0717C7E7AC45EE197

Previous Output: 2BF94CA75B69061C4440C606270AF37F2993782ED52FF80485BE221318465577
A10D27AB93BEA19188BEFF147038D723E356D4D5E9EEC420F1A64412A7A7FE6D

Signature: 8D99A4E546A9DA0CA8F88F1C28EA411CF086125CE9B9728DDE0997E3C7EC8C32
BA1CC764D745D372F79B5607E338003B3C1684CD1EA5972C06F1FB80AD363B77
BF63E186C6E514DB39FFF12E190234FE12C4494D44D3A9BFE175AF04560F99DD
1AA0724C5111ADC22260C6B8ADB4ECFCF0059266F673FA3F3EA02D404F505CCB
6338808944E22C510871037993123C9932EBDF7802F45FBE280146121688BCBE
08CE11DDE12B95CC57C028BDE2653C3612C754ACC2A7F8249F7E7EAC98C32354
E6C06C9909C707A597AE7D99951852F893F23167725347B5953E9416DE229F7E



Available today?!

- E.g.: NIST's randomness beacon
- One new value per minute



Certificate

NIST Beacon 1.0 used an X.509 certificate with the Federal Common Policy CA as the ultimate root authority for some of the records. The certificate is available [here](#)

The Beacon Signing Key changed in May of 2017. Not all records will successfully validate.

Viewer

The example application below uses the REST API described above to navigate the data.

To visit an arbitrary time value please click on the input control containing the date and time value to display a calendar to select the desired value.

First Start Chain Previous Next Last

12/26/2018 5:07 pm

Beacon Record

Version: Version 1.0

Frequency: 60 seconds

Time: 12/26/2018 5:07 pm (1545840420)

Seed Value: C2E5D63FA9B8276C1E4568BAE5B50C242856707519C295AEC71B99E72A0588CF
EE2D79A7BC7ADD71D8BAC87FF753730D8DBDC171CE2DF6D0717C7E7AC45EE197

Previous Output: 2BF94CA75B69061C4440C606270AF37F2993782ED52FF80485BE221318465577
A10D27AB93BEA19188BEFF147038D723E356D4D5E9EEC420F1A64412A7A7FE6D

Signature: 8D99A4E546A9DA0C8F88F1C28EA411CF086125CE9B9728DDE0997E3C7EC8C32
BA1CC764D745D372F79B5607E338003B3C1684CD1EA5972C06F1FB80AD363B77
BF63E186C6E514DB39FFF12E190234FE12C4494D44D3A9BFE175AF04560F99DD
1AA0724C5111ADC22260C6B8ADB4ECFCF0059266F673FA3F3EA02D404F505CCB
6338808944E22C510871037993123C9932EBDF7802F45FBE280146121688BCBE
08CE11DDE12B95CC57C028BDE2653C3612C754ACC2A7F8249F7E7EAC98C32354
E6C06C9909C707A597AE7D99951852F893F23167725347B5953E9416DE229F7E

Trustworthy?

Original Articles

Nonrandom Risk: The 1970 Draft Lottery

Starr Norton

Published online: 01 Dec 2017

[Download citation](#)
<https://doi.org/10.1080/10691898.1997.11910534>

[Full Article](#)
[Figures & data](#)
[References](#)
[Citations](#)
[Metrics](#)
[Licensing](#)
[PDF](#)

Abstract

The 1970 draft lottery for birthdates is reviewed as an example of a government effort at randomization whose inadequacy can be exhibited by a wide variety of statistical approaches. Several methods of analyzing these data – which were of life-and-death importance to those concerned – are given explicitly and numerous others are cited. In addition, the corresponding data for 1971 and for 1972 are included as are the alphabetic lottery data, which were used to select draftees by the first letters of their names. Questions for class discussion are provided. The article ends with a survey of primary and secondary sources in print.

Keywords: Chi-square, Classroom exercise, Correlation, Exploratory data analysis, Randomness, Regression

1. Introduction

1. Introduction

2 GUILTY OF BID TO RIG PENNSYLVANIA LOTTERY

By WILLIAM ROBBINS and SPECIAL TO THE NEW YORK TIMES MAY 21, 1981

About the Archive

This is a digitized version of an article from The Times's print archive, before the start of online publication in 1996. To preserve these articles as they originally appeared, The Times does not alter, edit or update them. Occasionally the digitization process introduces transcription errors or other problems. Please send any feedback to times-19th-century@times.co.uk.

Occasionally the digitization process introduces transcription errors or other problems. Please send reports of such problems to archive_feedback@nytimes.com.



Nick Perry, a popular Pittsburgh television personality, and a state official were convicted in a county court here today of a million-dollar attempt to rig the Pennsylvania lottery.

The two were also convicted of committing perjury when they appeared before a Philadelphia grand jury that was investigating the attempt.



Trustworthy?

Original Articles Nonrandom Risk: The 1970 Draft Lottery

Starr Norton

Published online: 01 Dec 2017

Download citation <https://doi.org/10.1080/1069188>

Full Article

Figures & data

Reference

Abstract

The 1970 draft lottery for birthdates is reviewed and its statistical properties are examined. The lottery, whose inadequacy can be exhibited by a wide variety of statistical approaches, is analyzed. The data – which were of life-and-death importance to those concerned – are given explicitly and numerous others are cited. In addition, the corresponding data for 1971 and for 1972 are included, as are the alphabetic lottery data, which were used to select draftees by the first letters of their names. Questions for class discussion are provided. The article ends with a survey of primary and secondary sources in print.

Keywords: Chi-square, Classroom exercise, Correlation, Exploratory data analysis, Randomness, Regression

1. Introduction

Good examples make a statistics course come alive, and are memorable afterwards. Among the case histories that have held up well over the years is the 1970 draft lottery, the data for which are still not

2 GUILTY OF BID TO RIG PENNSYLVANIA LOTTERY

By WILLIAM ROBBINS and SPECIAL TO THE NEW YORK TIMES MAY 21, 1981

About the Archive

This is a digitized version of an article from The Times's print archive, before the start of online publication in 1996. To preserve these articles as they originally appeared, The Times does not alter, edit or update them.

Occasionally the digitization process introduces transcription errors or other problems. Please send reports of such problems to archive_feedback@nytimes.com.

Nick Perry, a popular Pittsburgh television personality, and a state official were convicted in a county court here today of a million-dollar attempt to rig the Pennsylvania lottery.

The two were also convicted of committing perjury when they appeared before a Philadelphia grand jury that was investigating the attempt.

How to do better?
Design choices and tradeoffs!

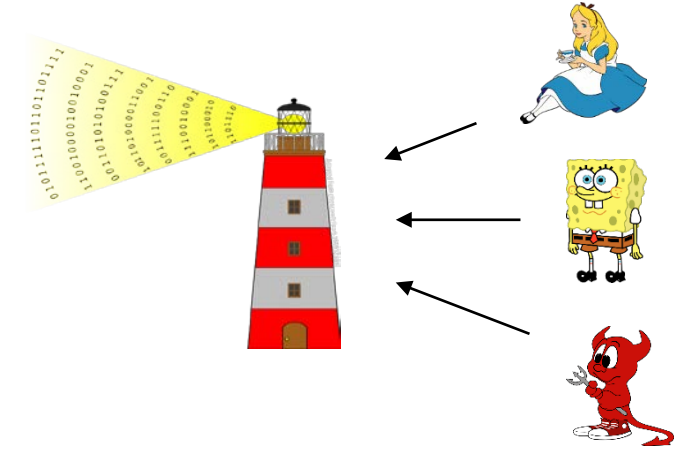


Design Choice 1: Input Sources

- *Private source*: e.g., NIST beacon
 - Potentially *high quality and high rate* randomness
 - But denies user access to inspect generation process
 - Requires trust: *not ok*
- *Publicly available data*: financial data, bitcoin block hashes, lottery data, weather (?)
 - Nice idea, *but*: sufficiently random? Rate?
- **User input**: outsource to the users, i.e., locally generated randomness
 - Users responsible to provide input!

Design Choice 1: Input Sources

- *Private source*: e.g., NIST beacon
 - Potentially *high quality and high rate* randomness
 - But denies user access to inspect generation process
 - Requires trust: *not ok*
- *Publicly available data*: financial data, bitcoin block hashes, lottery data, weather (?)
 - Nice idea, *but*: sufficiently random? Rate?
- *User input*: outsource to the users, i.e., locally generated randomness
 - Users responsible to provide input!



How random should it be?
As random *as they need!*



Design Choice 2: Beacon Operator

- *Autocratic collector*: e.g. run by a third party
 - Computation is blackbox, *no proof of honesty*
- *Specialized Multi-Party Computation (MPC)*: collectively produce randomness
 - ... typically from their own inputs
 - Despite significant work in the field, this approach is *difficult to scale*
 - *Addition or removal of a user* requires a new setup phase
 - But might fit in a controlled private context
- *Transparent authority model*: single entity collects inputs and publishes them
 - Focus on *transparency*: users can observe and verify the beacon



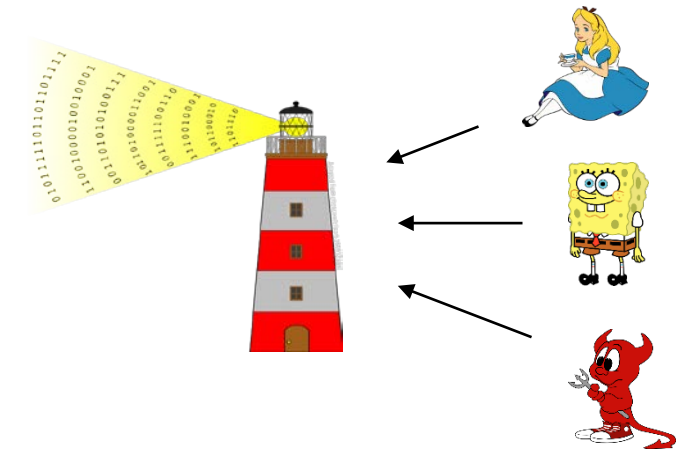
Design Choice 2: Beacon Operator

- *Autocratic collector*: e.g. run by a third party
 - Computation is blackbox, *no proof of honesty*
- *Specialized Multi-Party Computation (MPC)*: collectively produce randomness
 - ... typically from their own inputs
 - Despite significant work in the field
 - *Addition or removal* Byzantine behavior possible, but *difficult to hide!*
 - But might fit in a controlled private context
- *Transparent authority model*: single entity collects inputs and publishes them
 - Focus on *transparency*: users can observe and verify the beacon



Our Contributions

- A randomness beacon requiring *minimal trust*
 - Based on the *transparent authority* model which relies on *user input*
 - Beacon operator has *no private information*
- Allows users to
 - *Join anytime* and at low overhead
 - Make subtle decisions on *when to trust* the output
- Practical:
 - Prototype demonstrates *scalability* of our approach
 - Beacon can be deployed on distributed *ledger platforms*



Our Approach in a Nutshell

- No private information: all *inputs are hashed and released to the public* in batches before the computation
- Uses *commitments* and *verifiable delay function* to ensure that the operator *cannot try more than one commitment* before running out of time
- The beacon protocol also lets *users verify*.
 - *inclusion of their input* in the randomness generation (by *committing* to user inputs before starting the computation)
 - *correct calculation* of the random value from the provided inputs
- Several *practical optimizations* (e.g., for scalability)



A Deeper Dive

- *To support „choosable“ randomness:* a user's input rate is not limited (except for DoS)
- *To increase security:*
 - *Fast and transparent publishing:* input, output, and any data needed for verification needs to be published as soon as possible
 - *“Deterministic”:* any party can compute the randomness alongside the beacon operator
 - *Open:* anyone can contribute to the beacon to influence random generation
- *For scalability:* different channels for input and output



A Deeper Dive

- *To support „choosable“ randomness:* a user's input rate is not limited (except for DoS)
- *To increase security:*
 - *Fast and transparent publishing:* input, output, and any data needs to be published as soon as possible
 - *“Deterministic”:* any party can compute it alongside the beacon operator
 - *Open:* anyone can influence random generation
- *For scalability:* low market value of output: everyone can compute it!

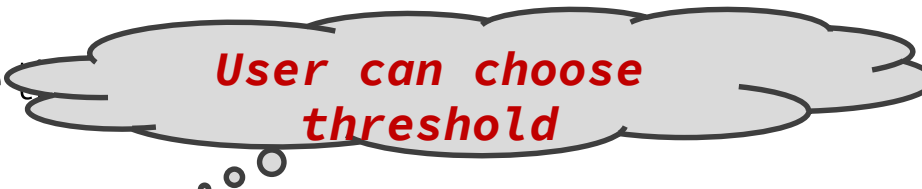


Delay Function

- Inherently *sequential functions* that require a given amount of time to run
 - No benefit from parallel execution
 - E.g., *sloth*, but also others
- Idea:
 - Cannot compute the delay function in the time between the users' input and the receiving the beacon's commitment to the input:
$$t_{\text{COMMITMENT}} - t_{\text{INPUT}} < T_{\text{DELAYFUNCTION}}$$
 - Operator cannot try more than one commitment before running out of time
 - But while hard to compute, *easy to verify*! E.g., *sequential hashing*.

Delay Function

- Inherently *sequential functions* that require a given amount of time to run
 - No benefit from parallel execution
 - E.g., *sloth*, but also others
- Idea:
 - Cannot compute the delay function in the time between receiving the beacon's commitment to the input:


 - $$t_{\text{COMMITMENT}} - t_{\text{INPUT}} < T_{\text{DELAYFUNCTION}}$$
 - Operator cannot try more than one commitment before running out of time
 - But while hard to compute, *easy to verify*! E.g., *sequential hashing*.

Prevents *input manipulation* and *last-draw attacks*.

Delay Function

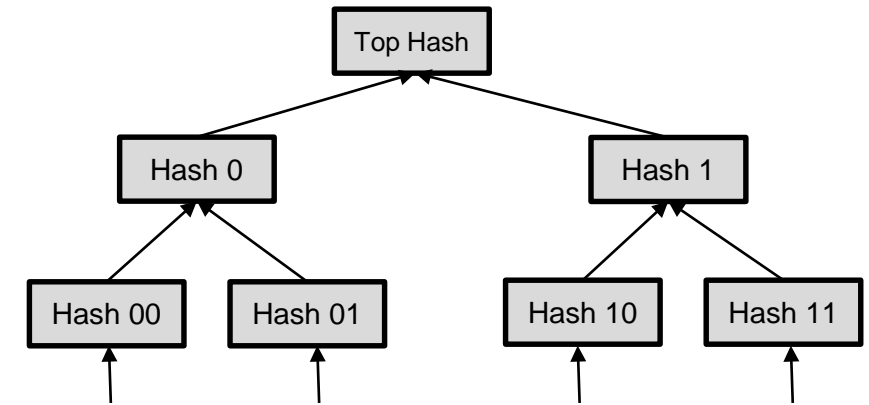
- Inherently *sequential functions* that require a given amount of time to run
 - No benefit from parallel execution
 - E.g., *sloth*, but also others
- Idea:
 - Cannot compute the delay function in the time between receiving the beacon's commitment to the input:

User can choose threshold

$$t_{\text{COMMITMENT}} - t_{\text{INPUT}} < T_{\text{DELAYFUNCTION}}$$
 - Operator cannot try more than one commitment before running out of time
 - But while hard to compute, *easy to verify*! E.g., *sequential hashing*.

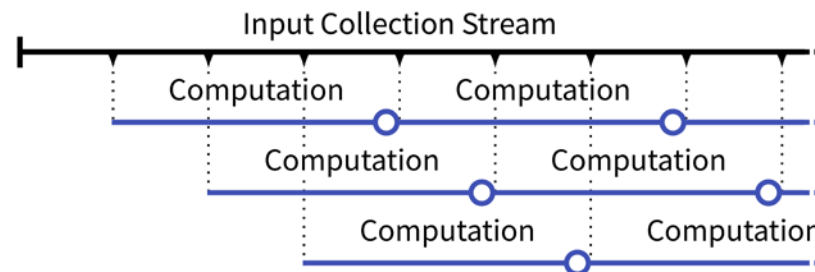
Combining Inputs

- During the input collection phase, users independently submit inputs to the beacon
- These inputs are aggregated into a **Merkle Tree**: the root is *input to the computation phase*
- This can also be used as commitment data that allows for *efficient verification of inclusion of a user's input* in logarithmic time



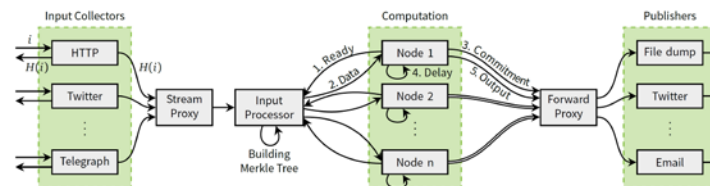
Pipelining

- Beacon operation: *input collection phase* followed by *computation phase*
- But: computation (delay function) takes much time, where users *cannot submit input*
- Solution: parallel beacon operation
 - Several delay functions *run in parallel*, but *offset* in time and on different input



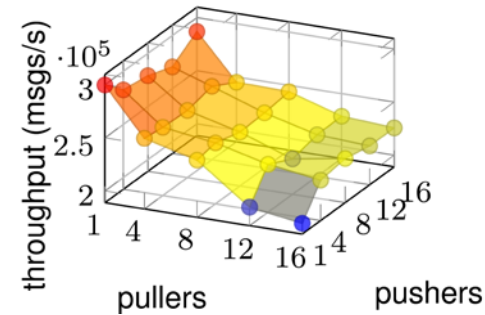
Proof-of-Concept Implementation

- Based on Python and ZeroMQ
- Two proxies:
 - *Forward proxy* between computation and publishers (forwards outputs, commitments, and proofs)
 - *Stream proxy* situated between input collectors and input processors
- Publicly available at: <https://github.com/randomchain/randbeacon>

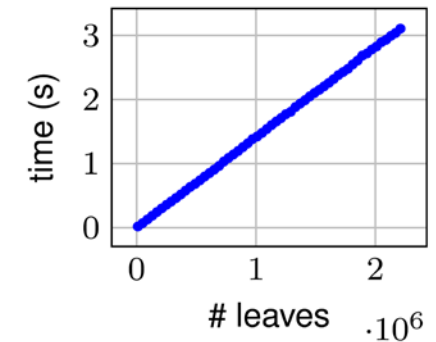


Evaluation

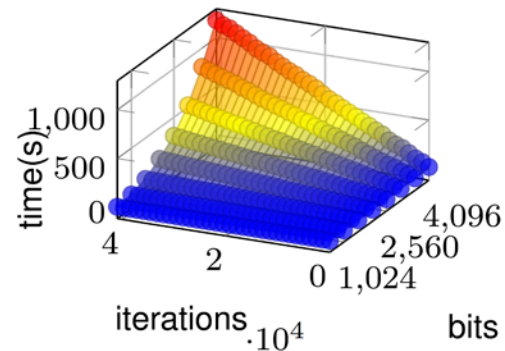
Stream proxy (bottleneck) can handle 1000s of msg/sec



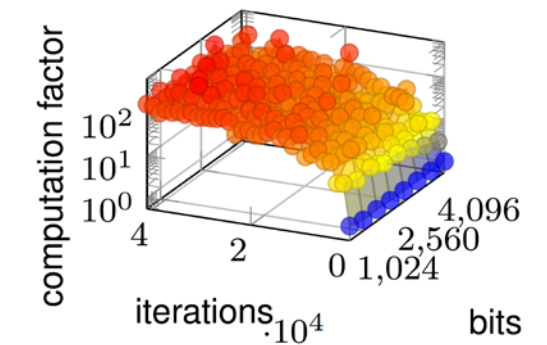
Building *Merkle tree* is fairly fast



Sloth computation time for different parameters



High asymmetry. computation time vs verification time





Distributed Ledger Implementation, e.g.: Lottery Application

- *Lottery*: the random value determines a winner randomly from the participants
- *Tradeoffs* how much of the beacon is *on-chain* (as a smart contract) and how much is *off-chain*
- 3 players:
 - *Owner* - Runs the lottery (e.g. smart contract owner)
 - *User* - Takes part in the lottery by sending a small payment to the lottery smart contract
 - *Beacon* - Beacon operator that provides a random value / lucky winner
- Goal of *lottery owner*: shave off some of the users' participation payments as a *reward*
- Users only participate if they trust random value provided by the beacon



Lottery: Implementation Options

	Delay Function on-chain	Delay Function off-chain, growing contract complexity			
Step \ Version	Full on-chain beacon	OFF	ITV	INP	OPT
Preprocessing	-	-	Beacon locks fund and nonce on-chain (to be released if not enough users participate within a certain time frame)	Like ITV	Like ITV
User Input	On-chain, together with lottery fee payment	Off-chain	Off-chain	On-chain, together with lottery fee payment	Like INP
Beacon Commitment	Not necessary	Off-chain. After users see their input committed in time, they send lottery fee payment	Users send Merkle root obtained off-chain (with beacon signature on root and nonce) with lottery fee payment to contract	Commitment is stored on-chain, if delivered timely and including all inputs in the commitment, else users are refunded and beacon loses fund	Like INP
Beacon Computation	On-chain	Off-chain, independent of lottery	Off-chain, after commitment is stored on-chain.	Like ITV	Like ITV
Beacon Output	On-chain	Store beacon value on-chain	Like OFF	Like OFF	Like OFF
Post-processing	-	User verify beacon and complain off-chain, may decide to not trust this lottery in the future (no influence on outcome of this draw)	On-chain verification. If verification is unsuccessful, beacon forfeits funds and users get lottery fees back	Like ITV	If evidence submitted by user, on-chain verification, if successful, users receive beacon funds and lottery fees, beacon forfeits funds
Reward	Owner and winning user receive rewards	Owner and winning user receive rewards	Owner, beacon and winning user receive rewards	Like ITV	Like ITV
Pros	Users do not have to worry about verification	Simple to implement, low gas consumption	Beacon compensated for its service	Beacon compensated, user only needs to interact with the contract	Beacon compensated, verification only executed on chain if someone complains
Cons	Requires many users to offset the on-chain computation cost, simpler and cheaper solutions without delay functions are possible for this scenario	Owner and user must know and adhere to timing of beacon, trust stems from incentives to repeat lottery execution. Beacon operator remunerated off-chain.	User interacts with off-chain beacon operator and smart contract. All honest users submit the same data. Verification executed on-chain for every draw	Verification executed on-chain for every draw, even though the beacon would typically be incentivised to be honest in this scenario	User must execute verification off-chain fast enough to react within the complaint window

Maximum Offchain

Lottery Implementation Options

	Delay	Function on-chain	Function off-chain, growing contract complexity		
Step \ Version	Full on-chain beacon	OFF	ITV	INP	OPT
Preprocessing	-	-	Beacon locks fund and nonce on-chain (to be released if not enough users participate within a certain time frame)	Like ITV	Like ITV
User Input	On-chain, together with lottery fee payment	Off-chain	Off-chain	On-chain, together with lottery fee payment	Like INP
Beacon Commitment	Not necessary	Off-chain. After users see their input committed in time, they send lottery fee payment	Users send Merkle root obtained off-chain (with beacon signature on root and nonce) with lottery fee payment to contract	Commitment is stored on-chain, if delivered timely and including all inputs in the commitment, else users are refunded and beacon loses fund	Like INP
Beacon Computation	On-chain	Off-chain, independent of lottery	Off-chain, after commitment is stored on-chain.	Like ITV	Like ITV
Beacon Output	On-chain	Store beacon value on-chain	Like OFF	Like OFF	Like OFF
Post-processing	-	User verify beacon and complain off-chain, may decide to not trust this lottery in the future (no influence on outcome of this draw)	On-chain verification. If verification is unsuccessful, beacon forfeits funds and users get lottery fees back.	Like ITV	If evidence submitted by user, on-chain verification, if successful, users receive beacon funds and lottery fees, beacon forfeits funds
Reward	Owner and winning user receive rewards	Owner and winning user receive rewards	Owner, beacon and winning user receive rewards	Like ITV	Like ITV
Pros	Users do not have to worry about verification	Simple to implement, low gas consumption	Beacon compensated for its service	Beacon compensated, user only needs to interact with the contract	Beacon compensated, verification only executed on chain if someone complains
Cons	Requires many users to offset the on-chain computation cost, simpler and cheaper solutions without delay functions are possible for this scenario	Owner and user must know and adhere to timing of beacon, trust stems from incentives to repeat lottery execution. Beacon operator remunerated off-chain.	User interacts with off-chain beacon operator and smart contract. All honest users submit the same data. Verification executed on-chain for every draw	Verification executed on-chain for every draw, even though the beacon would typically be incentivised to be honest in this scenario	User must execute verification off-chain fast enough to react within the complaint window

All beacon-related logic is *off-chain*. only the lottery logic is on-chain:

- The users *send inputs* to the beacon off-chain and *obtain commitment* off-chain
- Thereafter they *send the lottery payment* to the smart contract
- When the off-chain beacon value computation has finished, the lottery smart contract *fetches the value* and determines the *winner*
- Users can *verify* if the beacon matches the commitment and complain off-chain and decide not to trust this beacon in the future.

ITV

Lottery: Implementation Options

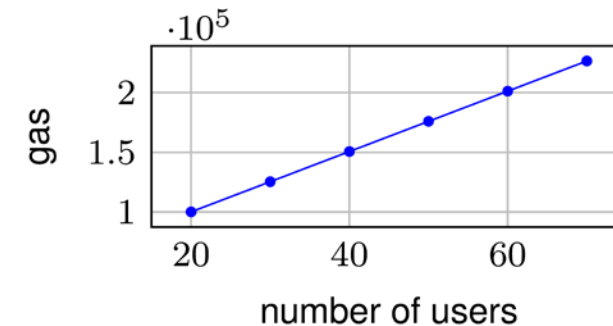
	Delay Function on-chain	Delay Function off-chain, growing contract complexity			
Step \ Version	Full on-chain beacon	OFF	ITV	INP	OPT
Preprocessing	-	-	Beacon locks fund and nonce on-chain (to be released if not enough users participate within a certain time frame)	Like ITV	Like ITV
User Input	On-chain, together with lottery fee payment	Off-chain	Off-chain	On-chain, together with lottery fee payment	Like INP
Beacon Commitment	Not necessary	Off-chain. After users see their input committed in time, they send lottery fee payment	Users send Merkle root obtained off-chain (with beacon signature on root and nonce) with lottery fee payment to contract	Commitment is stored on-chain, if delivered timely and including all inputs in the commitment, else users are refunded and beacon loses fund	Like INP
Beacon Computation	On-chain	Off-chain, independent of lottery	Off-chain, after commitment is stored on-chain.	Like ITV	Like ITV
Beacon Output	On-chain	Store beacon value on-chain	Like OFF	Like OFF	Like OFF
Post-processing	-	User verify beacon and complain off-chain, may decide to not trust this lottery in the future (no influence on outcome of this draw)	On-chain verification. If verification is unsuccessful, beacon forfeits funds and users get lottery fees back	Like ITV	If evidence submitted by user, on-chain verification, if successful, users receive beacon funds and lottery fees, beacon forfeits funds
Reward	Owner and winning user receive rewards	Owner and winning user receive rewards	Owner, beacon and winning user receive rewards	Like ITV	Like ITV
Pros	Users do not have to worry about verification	Simple to implement, low gas consumption	Beacon compensated for its service	Beacon compensated, user only needs to interact with the contract	Beacon compensated, verification only executed on chain if someone complains
Cons	Requires many users to offset the on-chain computation cost, simpler and cheaper solutions without delay functions are possible for this scenario	Owner and user must know and adhere to timing of beacon, trust stems from incentives to repeat lottery execution. Beacon operator remunerated off-chain.	User interacts with off-chain beacon operator and smart contract. All honest users submit the same data. Verification executed on-chain for every draw	Verification executed on-chain for every draw, even though the beacon would typically be incentivised to be honest in this scenario	User must execute verification off-chain fast enough to react within the complaint window

Adding beacon incentive and on-chain commitment (ITV):
Compensated beacon operator with the smart contract

- The beacon publishes its public key and a nonce and *locks some funds* in the smart contract
 - The beacon *loses* its locked funds *if verification fails*.
 - If the verification succeeds, the owner, beacon and winner *receive rewards*.
- The verification of the correct execution of sloth on the Merkle root is performed on-chain

Ethereum Smart Contract and Gas Cost

- Implemented an *Ethereum* smart contract
- Figure shows the *gas costs* for fetching the value from the beacon and drawing a winner for different number of users for the OFF model
- As expected, *linear increase* of the gas cost





Conclusions

- Design and implementation of a randomness beacon
 - Transparent and open: no trust needed
 - Beacon output can be publicly verified
- Performance study, also of the sloth protocol
- Lottery case study: realization tradeoffs
- Much *future work*! E.g., while nobody can bias the output, some parties may still know outcome a bit earlier than others.



universität
wien

Fakultät für Informatik

Thanks. Questions?
